SRI International

PLANNING

Technical Note 418

March 1987

By: Michael P. Georgeff
Program Director

Representation and Reasoning Program
Artificial Intelligence Center
Computer and Information Sciences Division
and
Center for the Study of Language and Information

**APPROVED FOR PUBLIC RELEASE:
DISTRIBUTION UNLIMITED**

This paper was originally published in *Annual Review of Computer Science, Volume 2*, J. F. Traub (ed), Palo Alto, California; Annual Reviews Inc., and is reproduced here with the permission of the publisher.

This research has been made possible by a gift from the System Development Foundation, by the Office of Naval Research under Contract N00014-85-C-0251, and by the National Aeronautics and Space Administration, Ames Research Center, under Contract NAS2-12521.

The views and conclusions contained in this paper are those of the author and should not be interpreted as representative of the official policies, either expressed or implied, of the Office of Naval Research, NASA, or the United States Government.

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
MAR 1987		**00-03-1987 to 00-03-1987**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Planning	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **SRI International,333 Ravenswood Avenue,Menlo Park,CA,94025**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**58**	

Contents

1 Introduction **1**

2 The Representation of Actions and Events **1**

2.1 Models of Action 2

2.2 The Situation Calculus 4

2.3 The STRIPS Representation 6

3 Plan Synthesis **9**

3.1 Plans 9

3.2 General Deductive Approaches 12

3.3 Planning as Search 15

3.4 Progression and Regression 16

3.5 Exploiting Commutativity 18

3.6 Improving Search 20

3.7 Hierarchical Planners 22

3.8 Other Planning Techniques 24

3.9 Planning and Scheduling 25

3.10 Operator Choice and Metaplanning 26

4 Multiagent Domains **27**

4.1 Action Representations 27

4.2 Causality and Process 30

4.3 Multiagent Planning 32

5 The Frame Problem **33**

6 Embedded Systems **36**

6.1 Execution Monitoring Systems 36

6.2 Reactive Systems 37

6.3 Rational Agents 39

1 Introduction

The ability to act appropriately in dynamic environments is critical to the survival of all living creatures. For lower life forms, it seems that sufficient capability is provided by stimulus-response and feedback mechanisms. Higher life forms, however, must be able to anticipate the future and form plans of action to achieve their goals. Reasoning about actions and plans can thus be seen as fundamental to the development of intelligent machines that are capable of dealing effectively with real-world problems.

Researchers in artificial intelligence (AI) have long been concerned with this area of investigation [73]. But, as with most of AI, it is often difficult to relate the different streams of research and to understand how one technique compares with others. Much of this difficulty derives from the varied (and sometimes confused) terminology and the great diversity of problems that arise in real-world planning. Indeed, there are few practical planning systems for which the class of appropriate applications can be clearly delineated.

This article attempts to clarify some of the issues that are important in reasoning about actions and plans. As the field is still young, it would be premature to expect us to have a stable foundation on which to build a discipline of planning. Nevertheless, I hope that the following discussion contributes toward that objective and that it will help the reader to evaluate the pertinent literature.

2 The Representation of Actions and Events

Humans spend a great deal of time deciding and reasoning about actions, some with much deliberation and some without any forethought. They may have numerous desires that they wish fulfilled, some more strongly than others. It is often necessary to accommodate conflicting desires, to choose among them, and to reason about how best to accomplish those that are chosen. This choice, and the means chosen to realize these ends, will depend upon currently held beliefs about present and future situations, and upon any committments or intentions that may have been earlier decided upon. Often it will be necessary to obtain more information about the tasks to be performed, either prior to choosing a plan of action or during its execution. Furthermore, our knowledge of the world itself is frequently incomplete, making it necessary for us to

have some means of forming reasonable assumptions about the possible occurrence of other events or the behaviors of other agents.

All this has to be accomplished in a complex and dynamic world populated with many other agents. The agent planning or deciding upon possible courses of action can choose from an enormous repertoire of actions, and these in turn can influence the world in exceedingly complicated ways. Moreover, because of the presence of other agents and processes, the environment is subject to continuous change – even as the planner deliberates on how best to achieve its goals.

2.1 Models of Action

To tackle the kind of problems mentioned above, we first have to understand clearly what entities we are to reason about. The traditional approach has been to consider that, at any given moment, the world is in one of a potentially infinite number of *states* or *situations*. A world state may be viewed as a snapshot of the world at a given instant of time. A sequence of world states is usually called a *behavior*; one that stretches back to the beginning of time or forward to its end is called a *world history* or a *chronicle*. Such a world history, for example, may represent the past history of the actual world, or some potential future behavior.

The world can change its state only by the occurrence of an *event* or *action*. An *event type* is usually modeled as a set of behaviors, representing all possible occurrences of the event in all possible world histories. Thus, the event type "John running around a track three times" corresponds to all possible behaviors in which John does exactly that – namely, run around *some* track during *some* interval at *some* location exactly three times. An *event instance* is a particular occurrence of an event type in a particular world history. However, where there is no ambiguity, we shall call event types simply events.

An *action* is a special kind of event, namely, one that is *performed* by some agent, usually in some intentional way. For example, a tree's shedding of its leaves is an event but not an action; John's running around a track is an action [in which John is the agent]. Philosophers make much of this distinction between actions and events, primarily because they are interested in activities that an agent decides upon, rather than those events that are not caused by the agent (such as leaves falling from a tree)

or that involve the agent in some unintentional way (such as tripping over a rug) [19]. For our purposes, however, we can treat these terms synonymously.

We shall begin by restricting our attention to domains in which there is no concurrent activity, as could be used to represent a single agent acting in a static environment. In these domains, it is only necessary to consider the initial and final states of any given event, as nothing can happen during the event to change its outcome. Consequently, an event can be modeled as a set of pairs of initial and final states, rather than as a set of complete behaviors. If, in addition, we limit ourselves to deterministic events, this relation between initial and final states will be functional; that is, the initial state in which an event occurs will uniquely determine the resulting final state. Of course, there may be certain states in which an event cannot be initiated; the set of states in which it *can* is usually called the *domain* of the event.

Events may be composed from other events in a number of ways. As they are just relations (or, in the more general case, sets of behaviors), the two simplest means of composition are set union and intersection. For example, the intersection of the event in which Mary hugs John and the event in which Mary kisses John is the event in which Mary both hugs *and* kisses John. We can also compose events sequentially; for example, to yield the event in which Mary first hugs John and then kisses him.

We also want to be able to say that certain *properties* hold of world states. For example, in some given state, it might be that a specified block is on top of some other block, or that its color is red. But what kind of entities are such properties? For example, consider the property of redness. In a static world, we might model this property as a set of individuals (or objects), namely, those that are red. However, in dynamic worlds, the individuals that are red can vary from state to state; we therefore cannot model redness in this way.

One approach is to view the properties of the domain as relating to particular states. Thus, instead of representing redness as a set of individuals, it is instead represented by a relation on individuals and states; a pair $\langle A, s \rangle$ would be a member of this relation just in case A were red in state s. Such entities are commonly called *situational relations.*

Another, more elegant way to handle this problem is to introduce the notion of a *fluent* [74], which is a function defined on world states. Essentially, a given fluent corresponds to some property of world states, and its value in a given state is the

value of that property in that state. For example, the property of redness could be represented by a fluent whose *value* in a given state is the set of individuals that are red in that state.

Fluents come in a variety of types. A fluent whose value in a given state is either *true* or *false* is usually called a *propositional* fluent. For example, the property of it being raining could be represented by a propositional fluent that has the value *true* in those states in which it is raining and the value *false* when it is not raining. An equivalent view of propositional fluents identifies them with the set of states for which the property is true.

Fluents can also represent individuals (such as agents and objects); their value in a given state will be some specific individual that exists in that state. For example, one may have a fluent representing Caesar; the value of this fluent in any state will be whoever happens to be the emperor of Rome at that point in time. Similarly, one can introduce fluents whose value in a given state is a relation over individuals, or fluents whose values are functions from individuals to individuals [79]. The former are typically called *relational fluents*, the latter *functional fluents.* [1]

2.2 The Situation Calculus

Of course, in any interesting domain, it is infeasible to specify explicitly the functions and relations representing events and fluents. We therefore need some formal language for describing and reasoning about them. (Those unfamiliar with logical formalism and its use in AI should refer to the article by Levesque on knowledge representation [65] and Hayes' beautifully clear exposition of naive physics [51].)

McCarthy [74] proposed a formal calculus of situations (states) which has become the classical approach to this problem. In the variant we describe here, the logical terms of the calculus are used to denote the states, events, and fluents of the problem domain. For example, the event term $puton(A, B)$ could be used to denote the action in which block A is placed on top of block B. Similarly, the fluent term $on(A, B)$ could designate the fluent representing the proposition that A is on top of B. We could also introduce other event terms to denote composite events and other fluent terms to

[1] A good introduction to many of these concepts, including some of the logics mentioned in Section 2.2, can be found in the first six chapters of reference [25]. Note, however, that fluents are therein called *intensions.*

denote the different kinds of fluents and various individuals (such as A and B) in the domain.

The predicates in this situation calculus are used primarily to make statements about the values of fluents in particular states. For propositional fluents, we shall use the expression $holds(f, s)$ to mean that the fluent f has value *true* in state s. For example, $holds(on(A, B), s)$ will be true if the fluent denoted by $on(A, B)$ has value *true* in state s; that is, if block A is on top of B in s. We can use other predicates and function symbols to describe the properties of other kinds of fluents [69,74,79].

We must also be able to specify the state transitions associated with any particular event in the problem domain. The usual way to do this is to introduce the term $result(e, s)$ to designate the state resulting from the performance of event e in state s. For example, $result(puton(A, B), s)$ denotes the state that results when the action $puton(A, B)$ is initiated in state s. We can also use the *result* function to characterize those states that are *reachable* by the agent from some given state. That is, for any state s and any performable action e, the state denoted by $result(e, s)$ will be reachable from s and, in turn, from any other state from which s is itself reachable.

The well-formed formulas of this situation calculus may also contain the usual logical connectives and quantifiers. With this machinery, we can now express general assertions about the effects of actions and events when carried out in particular situations. For example, we can express the result of putting block A on top of block B as follows:

$$\forall s \,.\, holds(clear(A), s) \wedge holds(clear(B), s) \supset holds(on(A, B), result(puton(A, B), s)).$$

This statement is intended to mean that if blocks A and B are initially clear, then after the action $puton(A, B)$ has been performed, block A will be on top of B.

One problem with the above approach is the apparently large number of axioms needed to describe what properties are *unaffected* by events. For example, if block B were known to be red prior to our placing block A upon it, we would not be able to conclude, on the basis of the previous axiom alone, that block B would still be red afterwards. To do so, we require an additional axiom stating that the movement of block A does not change the color of block B:

$$\forall s \,.\, holds(color(B, red), s) \supset holds(color(B, red), result(puton(A, B), s))$$

In fact, we would have to provide similar axioms for every property of the domain left unaffected by the action. These are called *frame axioms*; being forced to specify them is commonly known as the *frame problem* [50].

Various other logical formalisms have been developed for representing and reasoning about dynamic domains. The most common are the *modal logics*, which avoid the explicit use of terms representing world state. One type of modal logic, called *temporal logic*, introduces various *temporal operators* for describing properties of world histories [94]. For example, the fact that it will rain sometime in the future can be represented by the formula $\Diamond raining$. Here, the temporal operator $\Diamond$ represents the temporal modality "at some time in the future;" the formula $\Diamond\phi$ means that there exists some future state for which the formula ϕ is true. The use of temporal operators corresponds closely to the way *tense* is used in natural languages; thus, it is claimed, these logics provide a natural and convenient means for describing the temporal properties of given domains.

Process logics are another kind of modal logic in which explicit mention of state is avoided [87]. These logics are based on the same model of the world as described above, but introduce programs (or plans) as additional entities in the domain (see Section 3.1). *Dynamic logics* can be viewed as a special class of process logics that are concerned solely with the input-output behavior of programs [48]. Hence, these logics are concerned with binary relations on world states rather than with entire behaviors. Harel [47] provides a good review of dynamic and related logics.

2.3 The STRIPS Representation

The STRIPS representation of actions, originally proposed by Fikes and Nilsson [30], is one of the most widely used alternatives to the situation calculus. It was introduced to overcome what were seen primarily as computational difficulties in using the situation calculus to construct plans. The major problem was to avoid (1) the specification of a potentially large number of frame axioms, and (2) the necessity of having the planner consider these axioms in determining the properties that hold at each point in the plan.

In the STRIPS representation, a world state is represented by a set of logical formulas, the conjunction of which is intended to describe the given state. Actions or events are represented by so-called *operators*. An operator consists of a *precondition*,

an *add list*, and a *delete list*. Given a description of a world state s, the precondition of an operator is a logical formula that specifies whether or not the corresponding action can be performed in s, and the add and delete lists specify how to obtain a representation of the world state resulting from the performance of the action in s. In particular, the add list specifies the set of formulas that are true in the resulting state and must therefore be added to the set of formulas representing s, while the delete list specifies the set of formulas that may no longer be true and must therefore be deleted from the description of s. This scheme for determining the descriptions of successive states is called the *STRIPS rule*.

For example, the following STRIPS operator can be taken to represent the action that moves block A from location zero to location 1.

Precondition: $loc(A,0) \wedge clear(A)$
Add list: $\{loc(A,1)\}$
Delete list: $\{loc(A,0)\}$

Let's say that some world is described by the formulas $\{loc(A,0), clear(A), red(A)\}$. Given this set of formulas, it is possible (trivially in this case) to prove that the precondition holds, so that the operator is then considered applicable to this world description. The description of the world resulting from application of this operator is $\{loc(A,1), clear(A), red(A)\}$.

It is important to note that the formulas appearing in the delete list of an operator are not necessarily *false* in the resulting state; rather, the truth value of each of these formulas is considered unknown (unless it can be deduced from other information about the resulting state). Operators can also be parameterized and thus can represent a whole class of actions.

Although the operators in STRIPS are intended to describe actions that transform world states into other world states, they actually define *syntactic* transformations on *descriptions* of world states. STRIPS should thus be viewed as a form of logic and the STRIPS rule as a *rule of inference* within this logic. Given this perspective, it is necessary to specify the conditions under which the STRIPS rule is *sound*. That is, for each operator and its associated action, the formulas generated by application of the operator should indeed be true in the state resulting from the performance of the action. Surprisingly, only very recently has anyone attempted to provide such a semantics, though the importance of doing so has long been recognized.

The problem is that soundness is not possible to achieve if the STRIPS rule is allowed to apply to arbitrary formulas. One way around this difficulty is to specify a set of *allowable* formulas and require that only such formulas occur in world descriptions, operator add lists, and operator delete lists (although the preconditions of an operator can involve arbitrary formulas). Lifschitz [67] shows that such a system is sound if, for every operator and its associated action, (1) every allowable formula that appears in the operator's add list is satisfied in the state resulting from the performance of the action, and (2) every allowable formula that is satisfied in the state in which the action is initiated and does not belong to the operator's delete list is satisfied in the resulting state. The latter condition is commonly known as the *STRIPS assumption.*

As described, the STRIPS representation avoids the specification of frame axioms that state what properties are left unchanged by the occurrence of actions. Furthermore, the lack of frame axioms allows a planner to better focus its search effort. On the other hand, STRIPS is not nearly as expressive as the situation calculus [117]. In particular, the STRIPS representation compels us to include in an operator's delete list all allowable formulas that could possibly be affected by the action, even if the truth value of some of these could be deduced from other axioms. For example, even if we were given an axiom stating that when Fred dies he stops breathing, an operator representing the fatal shooting of Fred would nonetheless have to include in its delete list *both* effects of the shooting.

To overcome this difficulty, it is tempting to modify the STRIPS rule so that formulas that can be *proved* false in the resulting state need not be included in an operator's delete list. This leads to the *extended STRIPS assumption*, which states that any formula that is satisfied in the initiating state and does not belong to the delete list will be satisfied in the resulting state, *unless* it is inconsistent to assume so. Unfortunately, noone has yet provided an adequate semantics for such an approach [95].

Another alternative is to allow any kind of formula to appear in state and operator descriptions and to modify the STRIPS rule so that only a certain class of *basic* formulas is passed from state to state. The idea is that the truth value of a nonbasic formula in the state resulting from application of an operator cannot be determined using the STRIPS rule; it must instead be derived from other formulas that are true in that state. In this way, we can often simplify considerably the add lists of operators. (Some early attempts to implement this idea [28,29] were flawed [117] and later

implementations [121] appear to work only under certain restrictions.)

Yet another variant representation is described by Pednault [89]. Each action is represented by an operator that describes how performance of the action affects the relations, functions, and constants of the problem domain. As with the STRIPS representation, the state variable is suppressed and frame axioms need not be supplied. For a restricted but commonly occurring class of actions, the representation appears as expressive as the situation calculus.

3 Plan Synthesis

Plan synthesis concerns the construction of some plan of action for one or more agents to achieve some specified goal or goals, given the constraints of the world in which these agents are operating. In its most general form, it is necessary to take into account the various degrees to which the agents desire that their goals be fulfilled, the various risks involved, and the limitations to further reasoning arising from the real-time constraints of the environment. However, we shall begin with the simpler problem in which an agent's goals are consistent and all of the same utility; we shall disregard reasoning about the consequences of plan failure; and we shall not concern ourselves with real-time issues. (In philosophy, this kind of planning is commonly called *means-ends reasoning*, and is considered to be just one of the many components comprising rational activity [8,18,19].)

3.1 Plans

The essential component of a plan is that, when given to an agent or machine to perform or *execute*, it will produce some behavior. For example, a program for a computer is a particular kind of plan. Exactly what behavior occurs in a given situation will depend on the agent (or machine) that attempts to execute the plan, as well as on the environment in which that agent is embedded. In domains populated by more than one agent, plans may be assigned to a number of agents to execute cooperatively; such plans are often called *multiagent* plans.

The execution of a plan, of course, need not always be successful. Thus, there are at least two types of behavior associated with any plan: those that can be considered

successful, in that the agent manages to execute each part of the plan without failure, and those that are generated by the plan but that, for some reason or another, turn out to be unsuccessful. As a rule, an unsuccessful behavior is one in which the agent has executed part of the plan successfully but then failed to execute some subsequent step. In many applications, both kinds of behavior must be taken into consideration – one's choice of plans often depends on the likelihood of plan failure and its consequences [40].

Plans usually have a definite structure that depends on how the plan has been composed from more primitive components. The standard ways of composing plans include *sequencing* (resulting in sequential plans), *choice* (conditional plans), *iteration* (iterative plans), and *recursion* (recursive plans). One can also define *nondeterministic* operators that allow an arbitrary choice of which component to execute next and *parallel* operators to allow for concurrent activity.

The components of such composite plans are usually called *subplans*; the basic plan elements admitting of no further decomposition are called *atomic* or *primitive* plans (or, somewhat misleadingly, *atomic actions*). Plans can have other properties as well; which ones are considered important will often depend on the domain of application. For example, a plan may be of a certain type, or have an associated risk or likelihood of success.

It is not necessary that a plan be composed solely of atomic elements. For example, a plan for the evening might simply be to get dressed and then go to the theater, without it being specified how either of these activities should be accomplished. Similarly, we may fully detail each of the steps in a plan but leave unspecified the order in which they should be performed. Such plans are often called *partial* plans.

To reason about plans, we have to introduce additional *plan terms* into our formal language of events and actions. A plan term simply denotes a plan, in the same way that state terms, event terms, and fluent terms denote respectively states, events, and fluents. Furthermore, we need to introduce various function symbols and predicates to describe the allowed plan composition operators and any other properties we choose to ascribe to plans.

As mentioned above, one of the most important properties of a plan is the set of successful behaviors it generates. In the case that there is no concurrent activity, this can be reduced to considering simply the transformation from initial to final states.

Let's represent the fact that state s_2 results from the execution of plan p by agent M when initiated in state s_1 by the expression $generate(M, \mathrm{p}, s_1, s_2)$. With this, we can now describe the effects of plan execution and the properties of the various plan composition operators.

For example, for some particular agent M and plan p, we might have

$$\forall s_1, s_2 \,.\, holds(\phi, s_1) \supset (generate(M, \mathrm{p}, s_1, s_2) \supset holds(\psi, s_2))$$

That is, if p is executed by agent M in a state in which ϕ holds, at the completion of execution ψ will hold. Assuming a fixed agent, we shall write the above formula simply as $exec(\mathrm{p}, \phi, \psi)$.

Strictly speaking, plans are not actions. We might have some predicates that apply to plans (such as whether or not they are partial, conditional, or unreliable) but that clearly do not apply to actions. However, predicates such as *generate* (and *exec*) allow us to specify the relation between plans and the actions or behaviors they generate. Process logic and dynamic logic may be viewed as variant notations for describing the same relation. However, some authors [44,90] *equate* plans with the state transformations or events they generate. Used carefully, and in well-circumscribed problem domains, this can be the most frugal way to do things; in more general settings, however, it is restrictive and can lead to unnecessary confusion.

Finally, we need to consider what a *goal* is. Goals are important because they are the things that agents try to accomplish in executing their plans of action. In most of the early AI literature, a goal was considered to be simply some specified set of world states; an agent would be said to accomplish or achieve its goal if it managed to attain one of these states. For this reason, these goals are often called *end goals* or *goals of attainment*. Other researchers take goals to be some desired *transformation* from some set of possible initial states to some set of final states. This is typically the case in the area of automatic programming, in which one attempts to synthesize programs that meet certain input-output requirements.

However, real-world agents have goals that are more complex than these. For example, an agent may have a goal to *maintain* some condition over an interval of time, such as to remain in a position of power. Some goals of maintenance correspond to the *prevention* of some condition. For example, one might have the goal of preventing Congress from discovering how certain illegally obtained funds are distributed. There

may also be goals in which the properties of some final state are not particularly important, but where the intervening *sequence of activities* is – for example, the goal to call in at a Swiss bank prior to returning home. Thus, in general, we can consider a goal to be some set of state sequences or behaviors; an agent succeeds in achieving such a goal if the actual behavior of the world turns out to be an element of this set.

Because in the general case goals are just particular sets of behaviors, they may be composed in the same way that events are. For example, the goal to place the books on the bookshelf and the goal to place the cups on the table can be combined to form the single composite goal to place both the books on the bookshelf *and* the cups on the table. Goals can also be composed sequentially; in this case, the composite goal may be to *first* place the books on the bookshelf and *subsequently* place the cups on the table. The behaviors corresponding to the former composite goal, of course, may be different from those corresponding to the latter one. In the first case, the goal is to achieve a state in which both the books are on the bookshelf *and* the cups are on the table, but there is no requirement regarding the order in which these tasks should be performed. In the second case, the ordering of tasks is specified, but there is no requirement that the books remain on the bookshelf while the cups are placed on the table.

Goal descriptions are often viewed as *specifications* for a plan: they describe the successful behaviors that execution of the plan should produce. Goals of attainment can be specified by stating simply the conditions that should hold after execution of the plan. Thus, they can be adequately described using the language we introduced in Section 2 for describing the properties of world states. Goals of transformation can be similarly specified. For describing more general kinds of goal, however, we must be able to express properties of sequences of states or events. To do this, we need the more expressive formalisms developed for multiagent domains (see Section 4).

3.2 General Deductive Approaches

Given a formulation of actions and world states as described in Section 2, the simplest approach to planning is to prove – by means of some automatic or interactive theorem-proving system – the existence of a sequence of actions that will achieve the goal condition. More precisely, suppose that we have some end goal ψ and that the initial

state satisfies some condition ϕ. Then the theorem to be proved is

$$\forall s \,.\, holds(\phi, s) \supset \exists z \,.\, holds(\psi, z) \wedge reachable(z, s)$$

That is, we are required to prove that there exists a state z, reachable from s, in which the goal ψ holds, given that ϕ holds in the initial state s.

For example, a plan to clear a block A, given an initial state in which B is on top of A and C is on B, could be constructed by proving the theorem

$$\forall s \,.\, holds(on(C, B), s) \wedge holds(on(B, A), s) \supset \exists z \,.\, holds(clear(A), z) \wedge reachable(z, s)$$

If done carefully, the proof could lead to a solution of the form

$$\begin{aligned} \forall s \,.\quad & holds(on(C, B), s) \wedge holds(on(B, A), s) \supset \\ & holds(clear(A), result(puton(B, table), result(puton(C, table), s))) \end{aligned}$$

That is, if C is initially on B, which in turn is on A, then A will be clear in the state resulting from depositing C and then B on the table.

Green [44] was the first to implement this idea. As he observed, however, it is essential to have the theorem prover provide the right kind of constructive proof. For example, consider being faced with a choice of two doors, behind one of which is a ferocious lion and the other a young maiden. In trying to maximize your lifespan, a theorem prover may well suggest that you simply open the door behind which lies the young maiden. Unfortunately, you may only be able to ascertain the maiden's location after opening the door – too late for you but of little concern to the planning system. This difficulty arises because the sequence of actions constructed by the planner can be conditional on properties of *future* states; that is, on properties that the agent executing the plan is not in a position to determine.

Manna and Waldinger [69] show how many of these problems can be solved by reasoning about plans rather than actions. The planning technique they develop is based on the following scheme. For a goal condition ψ and initial state satisfying ϕ, we attempt to prove the theorem

$$\exists \mathbf{p} \,.\, exec(\mathbf{p}, \phi, \psi) \wedge executable(\mathbf{p})$$

The aim is to find a plan $\mathbf{p}$ that satisfies this theorem. For example, a solution to the problem given above would be

$$exec((\mathbf{puton(B,table)} \;;\; (\mathbf{puton(C,table)}), on(B, A) \wedge on(C, B), clear(A)) \,,$$

where the symbol ";" represents the sequential composition of plans. The requirement regarding executability is included to prevent the planner from returning trivial or nonexecutable plans; this requirement is usually left implicit.

Conditional plans are not difficult to construct in this framework. For example, consider that we can construct two plans p_1 and p_2 that satisfy respectively $exec(\mathrm{p}_1, \gamma \wedge \phi, \psi)$ and $exec(\mathrm{p}_2, \neg\gamma \wedge \phi, \psi)$. Then it is straightforward to show that the conditional plan $\mathrm{p} = \textbf{if } \gamma \textbf{ then } \mathrm{p}_1 \textbf{ else } \mathrm{p}_2$ satisfies $exec(\mathrm{p}, \phi, \psi)$. However, one must be careful in introducing conditionals as they can expand the search space of potential solutions considerably.

The construction of plans involving recursion is difficult. To handle recursion, we have to provide the theorem prover with an induction axiom. There are various kinds of induction axioms that one can use. Manna and Waldinger [69] use the principle of well-founded induction, that is, induction over a well-founded relation. This is a general rule that applies to many subject domains, but there are two difficulties. First, each domain will have its own well-founded relations, which must be specified explicitly. Second, it is often necessary to strengthen the goal constraint so as to have the benefit of a strong enough induction hypothesis to make things work out properly. With human intuition, it may not be difficult to formulate such strengthened goals, but even in simple cases the requisite strengthening seems to be beyond the capability of current theorem provers.

Iteration is often as problematic as recursion. It is not difficult to insert a *fixed* iterative subplan into a plan, provided that we have appropriate axioms describing its behavior. However, it is not a simple matter to *synthesize* an iterative subplan. One approach would be to first form a recursive plan (with all its attendant problems) and then transform this, if possible, into an iterative plan. Strong [107], for example, shows how to convert certain classes of recursive plans (programs) into iterative ones. This part is not difficult; the truly hard task is constructing the recursive plan to begin with.

However, in some cases, the synthesis of iterative plans is straightforward. For example, if a certain goal condition has to be satisfied for some arbitrary number of objects, it is often possible to construct a plan that accomplishes the goal for one of the objects and then iterate that plan over all the remaining objects.

Instead of the situation calculus, any of the modal logics discussed in Section 2.2 could alternatively be used to represent knowledge of the problem domain [62,99,108]. Unfortunately, for most domains of interest, it appears that we require the expressive power of first-order modal logics for which suitable theorem provers are currently unavailable. An exception to this is the work of Stuart [108]. He is concerned with the synchronization properties of plans, which can be adequately described using propositional modal logic.

3.3 Planning as Search

The basis of planning is to find, out of all the possible actions that an agent can perform, which, if any, will result in the world's behaving as specified by the goal conditions, and in what order these actions should occur. It can thus be viewed as a straightforward search problem: find some or all possible orderings of the agent's actions that would result in achieving the specified goal, given the constraints of the world in which the agent is embedded. The number of possible action orderings is equal to the factorial of the number of actions performable by the agent. This makes the *general* problem computationally intractable (or what is usually called NP-hard).

Thus, a considerable part of the research effort in planning has been directed to finding effective ways of reducing this search, either by formulating the problem in some appropriate way, restricting the class of problems that can be represented, or by careful choice in the manner in which potential plans are examined (see reference [65] for a discussion of similar issues).

There are two common ways of viewing plan search techniques. One is to perceive the process as searching through a space of world states, with the transitions between states corresponding to the actions performable by the agent. Another view is that the search takes place through a space of partial plans, in which each node in the search space corresponds to a partially completed plan. The latter view is the more general, as the first can be seen as a special case in which the partial plan is extended by adding a primitive plan element to either end of the current partial plan.

Thus, we can characterize most approaches to the planning problem as follows. Each node in the search space corresponds to some possibly partial plan of action to achieve the given goal. The search space is expanded by further elaborating some component of the plan formed so far. We shall call the ways in which plans can

be elaborated *plan specialization operators.*[2] The plan space can be searched with a variety of techniques, both classical and heuristic [84,112].

At each stage, a very general plan (which admits of a potentially large number of behaviors) is increasingly refined until it becomes very specific. As it becomes more specific, the set of behaviors that can potentially satisfy the plan becomes smaller and smaller. The process continues until the plan is specific enough to be considered a solution to the problem; usually this is taken to mean that the plan can be executed by some specified agent. Alternatively, we can view the whole process as continually refining the *specifications* for a plan; that is, by reducing progressively the original goal to some composition of subgoals.

A different approach to planning involves plan *modification* rather than plan specialization. Thus, beginning with a plan that only *approximates* the goal specifications, various *plan modification operators* are applied repeatedly in an attempt to improve the approximation. Unlike plan specialization, the problem with plan modification is that it is often difficult to determine whether or not a particular modification yields a plan that is any closer to a solution. In many cases, a combination of plan specialization and modification can be used effectively. Furthermore, many techniques that are framed in terms of plan modification can be recast as equivalent plan-specialization techniques and vice versa.

In the next few sections, we shall examine the more important of these specialized planning techniques. For many of these, there exists a corresponding deductive method whereby certain constraints are imposed on the order in which inferences are drawn [35,56,69].

3.4 Progression and Regression

Before we consider specific planning techniques, let us introduce some new terminology. Consider a primitive plan a that, for some conditions ϕ and ψ, satisfies $exec(\mathbf{a}, \phi, \psi)$. That is, we are guaranteed that, after initiation of a in a state in which ϕ holds, ψ will hold at the completion of execution. If ψ is the strongest condition for which we can prove that this holds, we shall call ψ the *strongest provable postcondition* of a

[2]Note that plan specialization operators are not the same as plan composition operators, although often there is a correspondence between the two. The former operators map partial plans into more specific plans; the latter map some tuple of plans into a composite plan.

with respect to ϕ. We can similarly define the *weakest provable precondition* of **a** with respect to ψ to be the weakest condition ϕ that guarantees that ψ will hold if **a** is initiated in a state in which ϕ holds. Analogously, we can apply these terms to actions as well as plans.

Let's now consider how we could form a plan to achieve a goal ψ, starting from an initial world in which ϕ holds. That is, we are required to find a plan **p** that satisfies $exec(\mathbf{p}, \phi, \psi)$. For any primitive plan element **a**, this condition will be satisfied for **p** if:

1. $\mathbf{p} = \text{NIL}$ and $\forall s \,.\, holds(\phi, s) \supset holds(\psi, s)$.

2. $\mathbf{p} = \mathbf{a}$; **q**, where **q** satisfies $exec(\mathbf{q}, \gamma, \psi)$ and γ is the strongest provable post-condition of **a**.

3. $\mathbf{p} = \mathbf{q}$; **a**, where **q** satisfies $exec(\mathbf{q}, \phi, \gamma)$ and γ is the weakest provable precondition of **a** and ψ.

4. $\mathbf{p} = \mathbf{q}_1; \mathbf{a}; \mathbf{q}_2$, where, for some γ_1 and γ_2, **a** satisfies $exec(\mathbf{a}, \gamma_1, \gamma_2)$, $\mathbf{q}_1$ satisfies $exec(\mathbf{q}_1, \phi, \gamma_1)$, and $\mathbf{q}_2$ satisfies $exec(\mathbf{q}_2, \gamma_2, \psi)$.

Case (1) simply says that, if the goal condition is already satisfied, we need not plan anymore, i.e., the empty plan (NIL) will do. Now consider case (2). Let's say that we are guaranteed that, if we execute some primitive plan **a** in a state in which ϕ holds, γ will be true in the resulting state. Thus, if the plan begins with the element **a**, the rest of the plan must take us from a state in which γ is true to one in which ψ is true. We can take γ to be any condition that is guaranteed to hold after the execution of **a** but, to spare ourselves from planning for situations that cannot possibly occur, it is best to take γ to be the strongest of these conditions. Thus, case (2) amounts simply to forward-chaining from the initial state and is usually called *progression.* Case (3) is similar to case (2), except that we chain backward from the goal. It is usually called *regression*; the condition γ is often called the *regressed goal.* Case (4) is tantamount to choosing a primitive plan element somewhere in the middle of the plan, then trying to patch the plan at either end. In fact, case (4) is a generalization of cases (2) and (3).

It is straightforward to construct a simple planner that uses these rules recursively to build a plan. Initially the planner starts with the fully unelaborated plan **p**, then specializes this plan recursively, applying rules (2), (3), or (4) until, finally, rule (1) can

be applied. Clearly, whether or not a solution is obtained will depend on the choice of rules and the choice of primitive plan elements at each step. The algorithm works for any plan or action representation, requiring only that we be able to determine plan (or action) postconditions and preconditions, as described above. For example, GPS [83] and STRIPS [30] use STRIPS-like action representations and rules (1) and (4), whereas Rosenschein [99] employs dynamic logic to describe the effects of actions and uses rules (1), (2), and (3).

Rather than specify the execution properties of primitive plans alone (using STRIPS or some other variant), it can also be useful to provide information on the execution of partial plans. For example, it may be that going out to dinner and then to the theater results in one being happy, irrespective of the way in which this plan is eventually realized. I have elsewhere [40] called this *procedural knowledge*, on the grounds that such facts describe properties regarding the execution of certain procedures or plans. Using the reactive planning system PRS (see Section 6.3), Lansky and I show how such procedural knowledge can be used effectively for handling quite complex tasks, such as fault diagnosis on the space shuttle [41] and the control of autonomous robots [42]. The plan operators used in NOAH [101], DEVISER [116], and SIPE [121] (see Section 3.6) also allow the representation of procedural knowledge, although in more restrictive forms.

3.5 Exploiting Commutativity

Unfortunately, simple progression and regression techniques are too inefficient to be useful in most interesting planning problems. But is there anything better we can do? In the worst case, the answer is no – we simply have to explore all possible action orderings. However, the real world is not always so unkind.

For example, consider that one has a goal of stacking cups on a table *and* putting books on a bookshelf. It is often possible to construct plans that satisfy each of these component goals without regard to the other, then to combine these plans in some way to achieve the composite goal. (This might not be the case, however, if the cups were initially on the bookshelf or the books strewn randomly atop the table.)

For any two goals, if a plan for achieving one goal does not interfere with a plan to achieve the other goal, we say that the two plans are *interference-free* [39] or *com-*

mutative [84,89] with respect to these goals.[3] If this condition holds, any interleaving of these plans will satisfy both goals [39,64]. Partitioning the problem-solving space in this way can lead to a substantial reduction in the complexity of the problem, as it reduces the number of action orderings that need be considered. The assumption that two given goals can be solved independently (i.e., that the plans constructed for each goal will be interference-free) is known as the *strong linearity* assumption [109].

Unfortunately, in many cases of interest, the plans that are produced for each subgoal turn out not to be interference-free. It may nevertheless be possible to construct plans under the strong linearity assumption and, should they interfere with one another, patch them together in some way so that the desired goals are still achieved. Some early planners (notably HACKER [109] and INTERPLAN [113]) adopted this approach: they would construct plans that are flawed by interference with one another and then try to fix them by reordering the operations in the plans. These systems backtrack when they find interference; they reorder a couple of subgoals and then start planning to achieve them in the new order. However, this can be inefficient and is somewhat restricted in the class of problems that can be solved [117].

Waldinger [117] developed a more general planning method that, in forming a plan for a particular subgoal, took account of the constraints imposed by the subplans for previously solved subgoals. That is, he first constructs a plan to achieve one of the subgoals, without regard to any others. He then tries to achieve the next subgoal while maintaining the constraints imposed by the first plan. In particular, the first subgoal and its regressed conditions, as obtained from the initial plan, become goals of maintenance that must be satisfied by the new plan. These goals of maintenance are usually called *protected* conditions.

Kowalski [56] and Warren [118] follow essentially the same approach as Waldinger. Kowalski also examines ways of improving the efficiency of planning by combining regression and progression in various ways. However, none of these methods is complete (that is, none guarantees to find a solution if one exists), primarily because they match the regressed goals with the initial state prematurely. This deficiency was corrected by Pednault, who presents perhaps the most advanced version of this technique [89].

[3] In fact, these two notions are not identical. Plans that are interference free with respect to particular goals need not be commutative, and vice versa.

The foregoing approaches can be used with any sufficiently expressive action representation. Warren restricts himself to a simple STRIPS-like representation. An extended form of the STRIPS assumption is embedded in both the Waldinger and Kowalski systems. This, however, is not dealt with adequately by either author, so that the semantics of their action representations is not made clear. For example, Waldinger simply states that he uses "a default rule stating that, if no other regression rule applies, a given relation is assumed to be left unchanged by [the] action." This leads to serious semantic difficulties [95]. Kowalski [56] introduces a predicate *Diff*, the truth value of which is determined according to the *syntactic* structure of the terms that appear as arguments. Again the meaning of this is unclear. Pednault [89], on the other hand, uses an extended form of the STRIPS representation that does have a well-defined semantics.

For techniques that aim to exploit commutativity, it is clearly desirable that any composite goals be decomposed into maximally independent subgoals. However, unless one introduces some notion of independence (see Section 4.2), such decompositions are not possible to determine. In fact, all existing planners decompose composite goals on the basis of their *syntactic* structure alone, rather than on any properties of the domain itself. The reason that this often appears to work is probably a result of the way the predicates chosen to represent the world reflect some kind of underlying independence that has not been made explicit. For example, given the composite goal of stacking cups on the table and books on the bookshelf, one possible decomposition is into two subgoals, one of which is to stack half the cups and half the books and the other to stack the remainder of the cups and books. Clearly, in most situations, this turns out to be a poor decomposition of the composite goal; that most planners don't make this decomposition is simply a result of fortuitous choice of domain predicates. However, there are some cases in which this decomposition actually *is* the best one; for example, when half the cups are stored together with half the books in one container and the rest of the cups and books in another container.

3.6 Improving Search

The techniques outlined above provide only a limited number of plan specialization and modification operators and apply only to those cases in which the goal descriptions (plan specifications) consist of a *conjunction* of simpler goal conditions. Furthermore,

these operators are such that their application always results in a linear ordering of primitive plan elements. They are thus often called *linear* planners.

However, the search can often be improved by deferring decisions on the ordering of plan elements until such decisions are forced; by that time, we could well have acquired the information we need to make a wise choice. This is the technique adopted by the so-called *nonlinear planners.* These planners allow the partial plans formed during the search to be arbitrary partial orders over plan elements. Thus, instead of arbitrarily choosing an ordering of plan elements, they are left unordered until some conflict is detected among them, at which time some ordering constraint is imposed to remove the conflict. Nonlinear planners therefore do not have to commit themselves prematurely to a specific linear order of actions and can get by with less backtracking than otherwise. Examples of such planners include NOAH [101], NONLIN [110], DEVISER [116], and SIPE [121].

Another way in which we can defer making decisions that, at some later stage, may have to be retracted, is to allow the individuals (objects) that appear in a plan to be partially specified. This can be achieved by accumulating *constraints* on the properties that these individuals must satisfy, and by deferring the selection of any particular individual as long as possible [105,121]. For example, we may know that, to perform a certain block-moving action, the block being moved must weigh under five pounds. Instead of selecting a particular block that meets this constraint, we instead just post that constraint on the value of the logical variable denoting the block. Later, if we discover that the block should be red as well, we can then attempt to select a block that meets both requirements. This technique can be easily implemented by associating a list of constraints with each such variable and periodically checking these constraints for consistency. Mutual constraints among variables may be similarly handled.

Efficiency can often be further improved by introducing additional plan specialization and plan modification operators. Most nonlinear planners provide a great variety of such operators. For a restricted class of problems, Chapman [13] and Pednault [89] furnish a complete set of plan specialization and modification operators, and prove soundness and completeness for their systems. They also provide a good review of the techniques and failings of other nonlinear planners. Chapman also analyzes the complexity of various classes of these planners. Of course, these planners are no more efficient than their linear counterparts in the worst case.

3.7 Hierarchical Planners

The major disadvantage of the planners discussed so far is that they do not distinguish between activities that are critical to the success of the plan and those that are merely details. As a result, these planners can get bogged down in a mass of minutiae. For example, in planning a trip to Europe, it is usually a waste of time to consider, prior to sketching an itinerary, the purchase of tickets or the manner of travelling to the airport.

The method of *hierarchical planning* is first to construct an abstract plan in which the details are left unspecified, and then to refine these components into more detailed subplans until enough of the plan has been elaborated to ensure its success. The advantage of this approach is that the plan is first developed at a level at which the details are not computationally overwhelming.

For an approach to be truly hierarchical, we require that the abstract planning space be a *homomorphic image* of the original (ground) problem. Let us assume we have some function g that maps partial plans in the ground problem space into partial plans in the abstract space. Also, for any plan composition operator C in the ground space, let C' be the corresponding composition operator in the abstract space. Then, if $\mathbf{p_1}$ and $\mathbf{p_2}$ are partial plans in the ground space, we require that

$$g(C(\mathbf{p_1}, \mathbf{p_2})) = C'(g(\mathbf{p_1}), g(\mathbf{p_2}))$$

This simply captures the fact that the abstraction should preserve the structure of the original problem.

These requirements ensure that if we find a solution at the ground level there will exist a corresponding solution at the abstract level. Furthermore, they ensure that any plan constructed at the abstract level will be composed of plan elements that can be solved *independently* of one another at the lower levels of abstraction. Indeed, it is in this manner that the complexity of the problem is factored. The method can be extended to multiple levels of abstraction in a straightforward way.

There have been a number of attempts to devise hierarchical planning systems. The system ABSTRIPS [100] induced appropriate homomorphisms by neglecting the predicates specifying details of the domain and only retaining the important ones. Thus, for example, in determining how to get from Palo Alto to London, the predicates

describing the possession of plane tickets or the relative location of, say, Palo Alto to San Francisco airport may be initially neglected. This simplifies planning the global itinerary, leaving the details of how to collect the plane tickets and how to travel from home to the airport to be determined at lower levels of abstraction. Rosenschein [99] describes another hierarchical planning method in which the homomorphism between levels is given by the relationship between primitive plan elements, at the one level, and elaborated plans at the next lower level – what Rosenschein describes as the relation of being "correctly implemented."

Unfortunately, the notion of hierarchical planning is confused in much of the AI literature. In particular, there is considerable misunderstanding about the requirement that the problems at lower levels of abstraction be *independently* solvable. This does *not* mean that the plans formed at the lower levels of abstraction must be independent of the features of other plans formed at these levels. The important point is that any features upon which these plans may mutually depend should be reflected at the higher abstraction levels (i.e., be retained by the homomorphic mapping g). This ensures that any interactions are taken into account in the abstract space and the lower levels can pursue their planning independently of one another. Of course, this approach only works if the number of features that have to be reflected at the higher levels is considerably smaller than the number of features that the lower levels must deal with in their planning.

For example, it may be that a certain stove can only accommodate two saucepans at a time. In planning the overall preparation of a meal, it is important that this information be retained. In this way, the plan formed at the abstract level can account for the potential interference between the cookings of various dishes, and the cookings themselves can then be separately and independently planned. The resource mechanism used in SIPE [121] serves exactly this purpose.

Most hierarchical planners (e.g., ABSTRIPS [100] and Rosenschein's hierarchical planner [99]) first form an abstract plan and only then consider construction of the lower level plans. Stefik [105], on the other hand, describes an approach in which the planning at the lower levels of abstraction may proceed before the abstract plan is fully elaborated. This is achieved by allowing the abstract plan to be partial with respect to the values of certain critical variables and letting the lower level plans post global constraints on these values (although he describes this process somewhat differently). In this way, planning need not proceed top-down – from the abstract space to the

lower-level spaces – but rather can move back and forth between levels of abstraction.

The plan specialization operators used in some of the nonlinear planners (such as NOAH [101], NONLIN [111], SIPE [121], and DEVISER [116]) also allow planning at higher levels of abstraction prior to concentrating on the finer details of a plan. However, these planners are *not* strictly hierarchical. The difficulty is that, even if all the subplans produced at lower levels of abstraction are successful (i.e., satisfy the plan constraints imposed by the higher levels of abstraction), it may still not be possible to combine them into a solution of the original problem [99]. Furthermore, the potential for global interactions is dependent on arbitrarily fine details of the lower level solutions, and the advantages of hierarchy are thus largely lost. There are also difficulties in the way that the STRIPS assumption manifests itself in planning at the higher levels of abstraction; Wilkins [122] discusses this problem in more detail and offers a solution.

3.8 Other Planning Techniques

The planning techniques we have discussed so far are all domain independent, in the sense that the representation schemes and reasoning mechanisms are general and can be applied to a variety of problem domains. However, in many cases, techniques specific to the application may be preferable. Many such specialized planning systems have been developed, mostly concerned with robotic control, navigation, and manipulation. For example, Gouzenes [43] has investigated techniques for solving collision-avoidance problems, Brooks [9] considers planning collision-free motions for pick-and-place operations, and Myers [82] describes a collision-avoidance algorithm suited to multiple robot arms operating concurrently. Krogh and Thorpe [58] have combined algorithms for path planning and dynamic steering control into a scheme for real-time control of autonomous vehicles in uncertain environments, and Kuan [59] describes a hierarchical method for spatial planning. Lozano-Perez [68] was one of the first researchers to demonstrate the utility of changes of representation in spatial planning. McDermott and Davis [77] describe an approach to path planning in uncertain surroundings, and Brooks [11] has devised a symbolic map representation whose primitives are suited to the task of navigation and that is explicitly grounded on the assumption that observations of the world are approximate and control is inaccurate.

In many applications, it is necessary to plan not only to achieve certain conditions

in the world but also to *acquire* information about the world. Relatively little work has been done in this area. Both Moore [80] and Morgenstern [81] propose to treat this problem by explicitly reasoning about the knowledge state of the planning agent. In such a framework, a test is viewed as an action that increases the knowledge of the agent, and can be reasoned about in the same way as other actions. In many cases, however, we can get by with a much simpler notion of a test. For example, Manna and Waldinger [69] allow a set of primitive tests to be specified directly and utilize these in forming conditional plans. Lansky and I [40] represent tests simply as plans that are guaranteed to fail if the condition being tested is false at the moment of plan initiation. In this manner, plans involving complex tests can be constructed without having to reason about the knowledge state of the planner.

Sometimes, one must be able to reason explicity about units of time and the expected duration of events. For example, in planning the purchase of a used car, it may be necessary to know that the bank closes at 3 pm, and that the time taken to travel to the bank is about 15 minutes, depending on traffic density. Furthermore, it is often necessary to reason about the probabilities of event occurrences and the likelihood of future situations. Some initial work in these areas can be found in references [7,22,24,33,116,119].

3.9 Planning and Scheduling

The problem of scheduling can be considered a special case of the planning problem. Essentially, one has a set of activities to carry out while satisfying some set of goal and domain constraints. Unlike the aforementioned planning problems, however, these activities are usually completely detailed, requiring no means-ends analysis. The question is how to put these activities together while maintaining whatever constraints are imposed by the problem domain. These constraints usually concern the availability of resources (such as a certain machine being available to perform one activity at a time) and timing requirements (such that a certain plan must be completed by a specified time). More often than not, the problem involves more than one agent and therefore is really a special case of the multiagent planning problem discussed in Section 4.

The class of problems considered by scheduling systems, moreover, often goes beyond those handled by traditional planning systems. For example, the domain constraints are often much richer than those considered by traditional planning systems.

In addition, one usually wants an optimal or nearly optimal solution. Furthermore, many of the goals and constraints are *soft*; that is, while it is desirable that they be met, they *can* be relaxed if necessary (at a certain cost). Thus, scheduling requires some kind of cost-benefit analysis to determine the optimal solution.

There are various standard mathematical techniques for solving scheduling problems. However, the combinatorics encountered makes them unsuitable for most real-world applications, especially when rescheduling is involved. On the other hand, there have been few attempts in AI to tackle this challenge. Fox [33] has developed a number of systems for handling a broad range of scheduling constraints, and has explored the use of special heuristics to constrain the search. As yet, however, the general principles underlying this work remain to be examined [32].

3.10 Operator Choice and Metaplanning

As more and more plan specialization operators are introduced, one is faced with the problem of when (i.e., in which order) to apply them. There are various ways of handling this problem. The simplest, and most common, is to specify some particular order and embed this directly in the planning algorithm. Another approach is to specify, for each operator, the conditions under which it should be used. The operators are then invoked (triggered) nondeterministically, depending on the current state of the planner. These techniques are usually called *opportunistic planners* [52].

Alternatively, one can consider the problem of how to go about constructing a plan (which operators should be expanded next, which goal to work on, how to decompose a goal, when to backtrack, how to make choices of means to a given end, etc.) as itself a problem in planning. One could thus provide rules describing possible planning methodologies and let the system determine automatically, for each particular case, the best way to go about constructing a plan. This kind of approach is often referred to as *metaplanning* [21,34,40,106,120]. The obvious problem with metaplanning is to ensure that the time saved in better constraining the search is not lost in reasoning about how to achieve this focus of effort.

Metaplanning can be viewed as describing or axiomatizing the process of planning. In its fullest generality, it could involve the synthesis of metalevel plans based on the properties of basic metalevel actions [106]. However, metaplanning is rarely used in this general way. Rather, the metalevel axiomatization usually serves as a convenient

way to describe various planning strategies to some generic planning system, without hardwiring them into the interpreter. Thus, for example, to have the system construct plans in some manner, one would simply provide the system with an axiomatization of that particular planning technique. In addition, we can in this way describe plan specialization and modification operators that are intended specifically for certain domains.

4 Multiagent Domains

Most real worlds involve dynamic processes beyond the control of an agent. Furthermore, they may be populated with other agents – some cooperative, some adversarial, and others who are simply disinterested. The single-agent planners we have been considering are not applicable in such domains. These planners cannot reason about actions that the agent has no control over and that, moreover, may or may not occur concurrently with what the agent is doing. There is no way to express nonperformance of an action, let alone to reason about it.

We therefore need to develop models of actions and plans that are different from those we have previously considered. We need theories of what it means for one action to interfere with another. Many interactions are harmful, leading to unforseen consequences or deadlock. Some are beneficial, even essential (such as lifting an object by simultaneously applying pressure from both sides). We should be able to state the result of the concurrence of two events or actions. We need to consider cooperative planning, planning in the presence of adversaries, and how to form contingency plans. In addition, we shall require systems capable of reasoning about the beliefs and intentions of other agents and how to communicate effectively both to exchange information and to coordinate plans of action. Furthermore, these systems will sometimes need to infer the beliefs, goals, and intentions of other agents from observation of their behaviors.

4.1 Action Representations

Multiagent domains are those having the potential for concurrent activity among multiple agents or other dynamic processes. The entities introduced in earlier sections – world states, histories, fluents, actions, events, and plans – can also form the basis for

reasoning in these domains. However, most of the simplifying assumptions we made for handling single-agent domains cannot be usefully employed here. In particular, it is not possible to consider every action as a relation on states, as the effects of performing actions concurrently depends on what happens *during* the actions [37,90]. For example, in a production line making various industrial components, it is important to know what machines are used during each activity so that potential resource conflicts can be identified.

In addition, we need more powerful and expressive formalisms for representing and reasoning about world histories. For example, we should be able to express environmental conditions such as "The bank will stay open until 3pm" and "If it rains overnight, it will be icy next morning." Similarly, we have to be able to reason about a great variety of goals, including goals of maintenance and goals satisfying various ordering constraints [90].

As before, we can take an event type to be a set of state sequences, representing all possible occurrences of the event *in all possible situations* [4,36,75,76,90]. Unlike single-agent domains, however, the set of behaviors associated with a given event must include those in which other events occur *concurrently* or *simultaneously* with the given event. For example, the event type corresponding to "John running around a track three times" must include behaviors in which other events (such as the launch of the space shuttle, it being raining, or John being accompanied by other runners) are occurring concurrently.

One possible approach is to approximate concurrent activity by using an interleaving approximation [39,37,88]. This renders the problem amenable to the planning techniques that are used for single-agent domains. However, it is not possible to model simultaneous events using such an approach, which limits its usefulness in some domains.

Another approach to reasoning about multiagent domains is to extend the situation calculus to allow reasoning about world histories and simultaneous events. I show elsewhere [36] how this can be done by introducing the notion of an atomic event, which can be viewed as an instantaneous transition from one world state to another. Atomic events cannot be modeled as functions on world state, as it would then be impossible for two such events to occur simultaneously (unless they had exactly the same effect on the world). Given this perspective, the transition relation of an atomic

event places restrictions on those world relations that are directly affected by the event but leaves most others to vary freely (depending upon what else is happening in the world). This is in contrast to the classical approach, which views an event as changing some world relations but leaving most of them unaltered.

I also introduce a notion of *independence* to describe the region of influence of given events. This turns out to be critical for reasoning about the persistence of world properties and other issues that arise in multiagent domains. Indeed, what makes planning useful for survival is the fact that we can structure the world in a way that keeps most properties and events independent of one another, thus allowing us to reason about the future without complete knowledge of all the events that could possibly be occurring.

McDermott [76] provides a somewhat different formalism for describing multiagent domains, although the underlying model of actions and events is essentially as described above. Perhaps the most important difference is that world histories are taken to be dense intervals of states, rather than sequences; that is, for any two states in any given world history, there always exists a distinct state that occurs between them. The aim of this extension is to allow reasoning about continuous processes and may also facilitate reasoning about hierarchical systems.

Allen and Pelavin [4,90] introduce yet another formalism based on a variation of this model of actions and events. The major difference is that fluents are viewed as functions on *intervals* of states, rather than as functions on states. Thus, in this formalism, $holds(raining, i)$ would mean that it is raining over the interval of time i, which might be, for example, some particular time period on some specific day. The aim is that, by using intervals rather than states, we obtain a more natural and possibly more tractable language for describing and reasoning about multiagent domains. A similar approach has been developed by Kowalski and Sergot [57].

Note that in these interval-based calculi, world states, per se, need not be included in the underlying model – indeed, intervals become the basic entities that appear in world histories. The ways intervals relate to one another are more complex than for states, however. Whereas, in a given world history, states can only either precede or succeed one another, intervals can also meet, contain, and overlap one another in a variety of ways. Some work on formalizing the interval calculus and its underlying models can be found in references [3,57,60,61,90,102,115].

Yet another approach is suggested by Lansky [62], who considers events as primitive and defines state derivatively in terms of event sequences. Properties that hold of world states are then restricted to being temporal properties of event sequences. For example, one might identify the property "waiting for service" with the condition that an event of type "request" has occurred and has not been followed by an event of type "serve". Lansky uses a temporal logic for expressing general facts about world histories and, in part, for reasoning about them also.

4.2 Causality and Process

One problem I have not yet addressed is the apparent complexity of the axioms that describe the effects of actions. For example, while it might seem reasonable to state that the location of block B is independent of the movement of block A, this is simply untrue, as everyone knows, in most interesting worlds. Whether or not the location of B is independent of the movement of A will depend on a host of conditions, such as whether B is in front of A, on top of A, atop A but tied to a door, and so on.

One way to solve this problem is by introducing a notion of *causality*.[4] Two kinds of causality suggest themselves: one in which an event causes the simultaneous occurrence of another event; the other in which an event causes the occurrence of a subsequent event. These two kinds of causality suffice to describe the behavior of any procedure, process, or device that is based on discrete (rather than continuous) events.

For example, we might have a causal law to express the fact that, whenever a block is moved, any block atop it and not somehow restrained (e.g., by a string tied to a door) will also move. According to this view, causation is simply a relation between atomic events that is conditional on the state of the world at the time of the events. Causation must also be related to the temporal ordering of events; for example, one would want to assume that an event cannot cause another event that precedes it. Various treatments of causality can be found in several sources [4,62,76,103]. Most of these view causality as a simple relation between events; Shoham [104], however, takes a radical view and defines causality in terms of the *knowledge state* of the agent. Wilkins [124] indicates

[4] Although it might appear that the notion of causality also arises in single-agent domains, there seems little point in developing a theory of causality suited to such worlds. The power of causal reasoning lies in describing the properties of dynamic environments in which the actions of an agent may affect or initiate actions by other agents or processes.

how causal laws could be used effectively in traditional planning systems.

It is often convenient to be able to reason about groups of causally interrelated events as single entities. Such groupings of events, together with the causal laws that relate them to one another, are usually called *processes* (although Allen [4] uses the term quite differently). For example, we might want to amalgamate the actions and events that constitute the internal workings of a robot, or those that pertain to each component in a complex physical system. A machine (or agent) together with a plan of action may also be viewed as a special kind of process.

Charniak and McDermott [14] examine how everday processes may be reasoned about. For example, they consider the problem of reasoning about the filling of a bath tub, and how we can infer that it will eventually fill up if the tap is turned on (and the plug not pulled). However, they do not indicate how to provide an adequate formalism for describing such processes, particularly with regard to their interaction with other, possibly concurrent processes.

Indeed, the strength of the concept of process derives from the way the interaction among events in different processes is strictly limited. Surprisingly, there has been little work in AI in this direction, although similar notions have been around for a long time. For example, Hayes [51] describes the situation in which two people agree to meet in a week. They then part, one going to London and the other remaining in San Francisco. They both lead eventful weeks, each independently of the other, and duly meet as arranged. To describe this using world states, we have to say what each of them is doing just before and just after each noteworthy event involving the other. As Hayes remarks, this is clearly silly.

One approach to this problem is to specify a set of processes and classify various events and fluents as being either internal or external with respect to these processes [36,63]. If we then require that there be no *direct* causal relationship between internal and external events, the only way the internal events of a given process can influence external events (or vice versa) is through indirect causation by an event that belongs to neither category. Within the framework of concurrency theory, these intermediary events (more accurately, event types) are often called *ports*. Processes thus impose causal boundaries and independence properties on a problem domain, and can thereby substantially reduce cominatorial complexity [36,63].

The identifiability of processes depends strongly on the problem domain. In stan-

dard programming systems (at least those that are well structured), processes can be used to represent scope rules and are fairly easy to specify. In complex physical systems, it is often the case that many of the properties of one subsystem will be independent of the majority of actions performed by other subsystems; thus these subsystems naturally correspond to processes as defined here. Lansky [63,62] gives other examples in which processes are readily specified. In other situations, such specification might be more complicated. Moreover, in many real-world situations, dependence will vary as the spheres of influence and the potential for interaction change over time [51].

If we are to exploit the notion of process effectively, it is also important to define various composition operators and to show how properties of the behaviors of the composite processes can be determined from the behaviors of their individual components. For example, we should be able to write down descriptions of the behaviors of individual agents and, from these descriptions, deduce properties of groups of agents acting concurrently. We should *not* have to consider the internal behaviors of each of these agents to determine how the group as a whole behaves. The existing literature on concurrency theory [53,78] provides a number of useful composition operators, though this area remains to be explored [36].

4.3 Multiagent Planning

Despite the variety of formalisms developed for reasoning about multiagent domains, relatively few planning systems have been fully implemented. Allen and Koomen [5] describe a simple planner, based on a restricted form of interval logic [4]. While this technique is effective for relatively simple problems, it is not obvious that the approach would be useful in more complex domains. Furthermore, the semantics of their action representation is unclear, particularly with respect to the meaning of concurrent activity. Kowalski and Sergot [57] also describe systems for reasoning about events based on interval calculi. Again, it appears that the class of problems that can be considered is limited; however, these techniques appear adequate for a large range of dynamic database applications.

Another issue concerns how separate plans can be combined in a way that avoids interference among the agents executing the plans. In such a setting, one could imagine a number of agents each forming their own plans and then, after communicating their intentions (plans) to one another or a centralized scheduler, modifying these to avoid

interference. To solve this problem, it is necessary to ascertain, from descriptions of the actions occurring in the individual plans, which actions could interfere with one another and in what manner [39]. After this has been determined, a coordinated plan that precludes such interference must then be constructed. This plan can be formed by inserting appropriate synchronization actions (interagent communications) into the original plans to ensure that only interference-free orderings will be allowed [37]. Stuart [108] formalized this approach and implemented a synchronizer based on techniques developed by Wolper and Manna [70].

As the above work shows, the notion of intending or committing to a course of behavior is an important cooperative principal. Some researchers have investigated how multiple agents can cooperate and resolve conflicts by reasoning locally about potential payoffs and risks [96,98]. Other research has focused on the use of global organizational strategies for coordinating the behavior of agents operating asynchronously [17]. Another interesting development is the work of Durfee, Lesser, and Corkhill [27] on distributed analysis of sensory information. They show that, by reasoning about the local plans of others, individual tracking agents can form partial global plans that lead to satisfactory performance even in rapidly changing environments. Reasoning about communication is discussed by Rosenschein [97] and Cohen and Levesque [16].

Lansky [62] has developed a multiagent planner that exploits causal independencies. Unlike the approaches described above, constraints between events have to be specified explicitly. However, the system accommodates a wide class of plan synchronization constraints. Also, the process of plan synchronization is not limited to a strategy of planning to separately achieve each component task and then combining the results. Instead, a more general, adaptable strategy is used that can bounce back and forth between local (i.e., single-agent) and global (multiagent) contexts, adding events where necessary for purposes of synchronization. Planning loci can be composed hierarchically or even overlap.

5 The Frame Problem

Although the so-called frame problem has been regarded as presenting a major difficulty for reasoning about actions and plans, there is still considerable disagreement over what it actually is. Some researchers, for example, see the problem as largely

a matter of combinatorics [74,95]; others view it as a problem of reasoning with incomplete information [76]; and yet others believe it relates to the difficulty of enabling systems to notice salient properties of the world [49]. I shall take the problem to be simply that of constructing a formulation in which it is possible to readily specify and reason about the properties of events and situations. This gives rise to at least five related subproblems, which I discuss below.

The first of these is what I shall call the *combinatorial problem.* While it does not appear too difficult to give axioms that describe the *changes* wrought by some given action, it seems unreasonable to have to write down axioms describing all the properties *unaffected* by the action. Axioms of the latter kind are usually called *frame axioms* [74] and, in general, they need to be given (or, at least, be deducible) for all property-action pairs. (Note that I use "unaffected" rather than "unchanging." This is an important distinction if we want to allow for concurrent events, as in most real-world domains.) The real problem here is to avoid *explicitly* writing down (or having to reason with) all the frame axioms for every property-action pair. Most solutions to this problem attempt to formalize some closed-world assumption regarding the specification of these dependencies.

As McCarthy was the first to observe [72], two further problems arise as a result of the fact that specifying the effects of actions is usually subject to qualification. The first sort of qualification has to do with the conditions under which the action effects certain *changes* in the world. This is what I call the *precondition qualification* problem (also variously called the intra-frame problem [103] and, simply, the qualification problem [72]). The second sort of qualification concerns the extent of influence of the action (or what remains *unaffected* by the action). I call it the *frame qualification* problem (also called the inter-frame problem [103] and the persistence problem [76]). Let me give examples of these two kinds of qualification.

First, consider that we are trying to determine what happens if Mary fires a loaded gun [at point-blank range] at Fred [46]. Given such a scenario, we should be able to derive, without having to state a host of qualifications, that Fred dies as a result of the shooting. However, if we then discover that (or are given an extra axiom to the effect that) the gun was loaded with a blank round, the conclusion (that Fred dies) should be defeasible – i.e., we should be able to accommodate the notion of Fred's possibly being alive after the firing. This is the precondition qualification problem. Most solutions to this problem aim to formalize the rule: "These are the only *preconditions* that matter

as far as the performance of the action is concerned, *unless* it can be shown otherwise."

As an example of the frame qualification problem, consider the point at which Mary loads the gun prior to firing it at Fred. All things being equal, it should be possible to derive that Fred's state of being is unaffected by loading the gun. However, if we discover that Fred actually died while the gun was being loaded, it should be possible (without changing the theory, except for the additional axiom about the time of Fred's death) to accommodate this without inconsistency and, if desired, to derive other theorems about the resulting situation. Most solutions to this problem attempt to formalize the rule thus: "These are the only *effects* of the action (given the preconditions), *unless* it can be shown otherwise." Solutions to this last problem often provide a solution to the combinatorial problem; the two problems, however, are clearly distinct.

The fourth problem concerns the ability to write down certain axioms of invariance regarding world states. Following Finger [31], I shall call this the *ramification problem.* For example, I should be able to formulate an axiom stating that everyone stops breathing when they die. Then I should be able to state that the effect of shooting a loaded gun at Fred results in his death without having to specify that it also results in cessation of his breathing. This latter effect of the shooting action should be inferable from the first axiom describing the consequences of dying. This seems straightforward, but a problem arises from the fact that such axioms can complicate the solution of the previous problems.

The fifth problem, which I call the *independence problem*, arises primarily in multi-agent domains. In a dynamic world, or one populated with many agents, we want our solutions to allow for the independent activities of other agents. Most importantly, we do not want to have to specify explicitly all the external events that might occur. But if we leave the occurrence of such events unspecified, we do not want a solution to the previously mentioned problems to overcommit us to a world in which these events are thereby assumed *not* to have occurred.

There are a great variety of approaches to these problems, including the use of default logics [95]; nonmonotonic logics [76]; consistency arguments [24]; minimization of the effects of actions [66], abnormalities [71], event occurrences [38], and ignorance [103]; and some ad hoc devices [50]. All can be viewed as metatheories regarding the making of appropriate *assumptions* about the given problem domain [93]. This kind of

reasoning will necessarily be *nonmonotonic*: in the light of additional evidence, some assumptions may need to be withdrawn, together with any conclusions based on those assumptions.

There are, however, serious difficulties in providing an acceptable semantics for many of these approaches [50,95] and many don't yield the intended results [46]. Furthermore, it is not clear how to implement most of these schemes efficiently. Reference [12] contains a collection of the most recent papers regarding this issue.

6 Embedded Systems

Of course, the ability to plan and reason about actions and plans is not much help unless the agent doing the planning can survive in the world in which it is embedded. This brings us to perhaps the most important and also most neglected area of planning research – the design of systems that are actually *situated* in the world and that must operate effectively given the real-time constraints of their environment.

6.1 Execution Monitoring Systems

Most existing architectures for embedded planning systems consist of a plan constructor and a plan executor. As a rule, the plan constructor plans an entire course of action before commencing execution of the plan [30,116,123]. The plan itself is usually composed of primitive actions – that is, actions that are directly performable by the system. The rationale for this approach, of course, is to ensure that the planned sequence of actions will actually achieve the prescribed goal. As the plan is executed, the system performs the primitive actions in the plan by calling various low-level routines. Usually, execution is monitored to ensure that these routines achieve the desired effects; if they do not, the system may return control to the plan constructor so that it can modify the existing plan appropriately.

Various techniques have been developed for monitoring the execution of plans and replanning upon noticing potential plan failure [30,123]. The basis for most of these approaches is to retain with the plan an explicit description of the conditions that are required to hold for correct plan execution. Throughout execution, these conditions are periodically checked. If any condition turns out to be unexpectedly false, a replanning

module is invoked. This module uses various plan modification operators to change the plan, or returns to some earlier stage in the plan formation process and attempts to reconstruct the plan given the changed conditions.

However, in real-world domains, much of the information about how best to achieve a given goal is acquired during plan execution. For example, in planning to get from home to the airport, the particular sequence of actions performed depends on information acquired on the way – such as which turnoff to take, which lane to get into, when to slow down and speed up, and so on. Traditional planners can only cope with this uncertainty in two ways: (1) by building highly conditional plans, most of whose branches will never be used, or (2) by leaving low-level tasks to be accomplished by fixed primitive operators that are themselves highly conditional (e.g., the intermediate level actions (ILAs) used by SHAKEY [85]). The first approach only works in limited domains – the environment is usually too dynamic to anticipate all possible contingencies. The second approach simply relegates the problem to the primitive operators themselves, and does not provide any mechanism by which the higher-level planner can control their behavior.

To overcome this problem, at least in part, there has been some work on developing planning systems that interleave plan formation and execution [20,26]. Such systems are better-suited to real worlds than the kind of systems described above, as decisions can be deferred until they *have* to be made. The reason for deferring decisions is that an agent can only acquire *more* information as time passes; thus, the quality of its decisions can only be expected to improve. Of course, there are limitations resulting from the need to coordinate activities in advance and the difficulty of manipulating large amounts of information, but some degree of deferred decision-making is clearly desirable.

6.2 Reactive Systems

Real-time constraints pose yet further problems for traditionally structured systems. First, the planning techniques typically used by these systems are very time consuming. While this may be acceptable in some situations, it is not suited to domains where replanning is frequently necessary and where system viability depends on readiness to act. In real-world domains, unanticipated events are the norm rather than the exception, necessitating frequent replanning. Furthermore, the real-time constraints of

the domain often require almost immediate reaction to changed circumstances, allowing insufficient time for this type of planning.

A second drawback of traditional planning systems is that they usually provide no mechanisms for responding to new situations or goals during plan execution, let alone during plan formation. Indeed, the very survival of an autonomous system may depend on its ability to react quickly to new situations and to modify its goals and intentions accordingly. These systems should be able to reason about their current intentions, changing and modifying these in the light of their possibly changing beliefs and goals. While many existing planners have replanning capabilities, none have yet accommodated modifications to the system's underlying set of goal priorities.

Finally, traditional planners are overcommitted to the planning strategy itself – no matter what the situation, or how urgent the need for action, these systems *always* spend as much time as necessary to plan and reason about achieving a given goal before performing any external actions whatsoever. They do not have the ability to decide when to stop planning, nor to reason about the trade-offs between further planning and longer available execution time. Furthermore, these planners are committed to a single planning technique and cannot opt for different methods in different situations. This clearly mitigates against survival in the real world.

Even systems that interleave planning and execution are still strongly committed to achieving the goals that were initially set them. They have no mechanisms for changing focus, adopting different goals, or reacting to sudden and unexpected changes in their environment.

A number of systems developed for the control of robots have a high degree of reactivity [1,2]. Even SHAKEY [85] utilized reactive procedures (ILAs) to realize the primitive actions of the high-level planner (STRIPS), and this idea is pursued further in some recent work by Nilsson [86]. Another approach is advocated by Brooks [10], who proposes decomposition of the problem into *task-achieving* units in which distinct behaviors of the robot are realized separately, each making use of the robot's sensors, effectors, and reasoning capabilities as needed. This is in contrast to the traditional approach in which the system is structured according to *functional* capabilities, resulting in separate, self-contained modules for performing such tasks as perception, planning, and task execution. Kaelbling [55] proposes an interesting hybrid architecture based on similar ideas.

Such architectures could lead to more viable and robust systems than the traditionally structured systems. Yet most of this work has not addressed the issues of general problem-solving and commonsense reasoning; the work is instead almost exclusively devoted to problems of navigation and execution of low-level actions. It remains to extend or integrate these techniques with systems that have the ability to completely change goal priorities, to modify, defer, or abandon current plans, and to reason about what is best to do in light of the current situation.

6.3 Rational Agents

Another promising approach to providing the kind of high-level goal-directed reasoning capabilities, together with the reactivity required for survival in the real world, is to consider planning systems as rational agents that are endowed with the psychological attitudes of belief, desire, and intention. The problem that then arises is specifying the properties we expect of these attitudes, the ways they interrelate, and the ways they determine rational behavior in a situated agent.

The role of beliefs and desires in reasoning about action has a long history in the philosophical literature [18]. However, only relatively recently has the role of intentions been carefully examined [8]. Moreover, there remain some major difficulties in formalizing these ideas. One serious problem is simply to choose an appropriate semantics for these notions. In particular, it is important to take into account the fact that the beliefs, desires, and intentions of an agent are *intensional* objects rather than *extensional* ones. For example, someone who does not know that the President of the United States is Ronald Reagan may well desire to meet one but not the other. Halpern [45] provides an excellent collection of papers on reasoning about knowledge and belief. The most serious attempt to formalize some basic principles governing the rational balance among an agent's beliefs, intentions, and consequent actions can be found in the work of Cohen and Levesque [15].

Lansky and I have been largely concerned with means-ends reasoning in dynamic environments, and with the way partial plans affect practical reasoning and govern future behavior [41]. We have developed a highly reactive system, called PRS, to which is attributed attitudes of belief, desire, and intention. Because these attitudes are explicitly represented, they can be manipulated and reasoned about, resulting in complex goal-directed and reflective behaviors. The system consists of a *data base*

containing current *beliefs* or facts about the world, a set of current *goals* or *desires* to be realized, a set of *procedures* or *plans* describing how certain sequences of actions and tests may be performed to achieve given goals or to react to particular situations, and an *interpreter* or *reasoning mechanism* for manipulating these components. At any moment, the system also has a *process stack*, containing all currently active plans, which can be viewed as the system's current *intentions* for achieving its goals or reacting to some observed situation.

The set of plans includes not only procedural knowledge about a specific domain, but also *metalevel* plans—that is, information about the manipulation of the beliefs, desires, and intentions of the system itself. For example, a typical metalevel plan would supply a method for choosing among multiple relevant plans, for achieving a conjunction of goals, or for deciding how much more planning or reasoning can be undertaken, given the real-time constraints of the problem domain.

The system operates by first forming a partial overall plan, then figuring out near-term means, executing any actions that are immediately applicable, further expanding the near-term plan, executing further, and so on. At any time, the plans the system intends to execute (i.e., the selected plans) are structurally partial—that is, while certain general goals have been decided upon, specific questions about the means to attain these ends are left open for future reasoning.

Furthermore, not all options that are considered by the system arise as a result of means-end reasoning. Changes in the environment may lead to changes in the system's beliefs, which in turn may result in the consideration of new plans that are not means to any already intended end. For example, the system may decide to drop its current goals and intentions completely, adopting new ones in their stead. This ability is vital in worlds in which emergencies of various degrees of severity can occur during the performance of other, less important tasks.

While the above work attempts to show how means-ends reasoning may be accomplished by systems situated in real-world environments, little research has been done in providing theories of *decision making* that are appropriate to resource-bounded agents. Researchers in philosophy, as well as decision theory, have long been concerned with the question of how a rational agent weighs alternative courses of action [54]. This work has largely assumed, either explicitly or implicitly, idealized agents with unbounded computational resources. In reality, however, agents do not have arbitrarily long to

decide how to act, for the world is changing around them while they deliberate. If deliberation continues for too long, the very beliefs and desires upon which deliberation is based, as well as the real circumstances of the action, may change. These and related issues are explored by Bratman [8] and Thomason [114]. Dean [23] discusses some methods whereby a planning system can recognize the difficulty of the problems it is attempting to solve and, depending on the time it has to consider the matter and what it stands to gain or lose, produce solutions that are reasonable given the circumstances.

Systems that are situated in worlds populated with other agents also have to be able to reason about the behaviors and capabilities of these other systems. This requires complex reasoning about interprocess communication [6,16], and the ability to infer the beliefs, goals, and intentions of agents from observations of their behavior [91,92]. The challenge remains, however, to design situated planning systems capable of even the simplest kinds of rational behavior.

Acknowledgments

I wish to thank particularly Amy Lansky, Nils Nilsson, Martha Pollack, and Dave Wilkins for their critical readings of earlier drafts of this paper. I am also indebted to Margaret Olender for her patient preparation of the bibliography and to Savel Kliachko for his erudite editorial advice.

References

[1] J. S. Albus. *Brains, Behavior, and Robotics.* McGraw-Hill, Peterborough, New Hampshire, 1981.

[2] J. S. Albus, A. J. Anthony, and R. N. Nagel. Theory and practice of hierarchical control. In *Proceedings of the Twenty-Third IEEE Computer Society International Conference*, 1981.

[3] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, 1982.

[4] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[5] J. F. Allen and J. A. Kooman. Planning using a temporal world model. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 741–747, Karlsruhe, West Germany, 1983.

[6] D. E. Appelt. Planning English referring expressions. *Artificial Intelligence*, 26:1–34, 1985.

[7] C. E. Bell and A. Tate. Using temporal constraints to restrict search in a planner. In *Proceedings of the Third Workshop of the Alvey IKBS Programme Planning Special Interest Group*, Sunningdale, Oxfordshire, UK, 1985.

[8] M. Bratman. *Intention, Plans, and Practical Reason.* Harvard University Press, Cambridge, Massachusetts, forthcoming.

[9] R. A. Brooks. Planning collision-free motions for pick-and-place operations. *International Journal of Robotics Research*, 2(4):19–40, 1983.

[10] R. A. Brooks. *A Robust Layered Control System for a Mobile Robot.* Technical Report 864, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1985.

[11] R. A. Brooks. Visual map making for a mobile robot. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 824–829, St. Louis, Missouri, 1985.

[12] F. Brown. *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop.* Morgan Kaufmann, Los Altos, California, 1987.

[13] D. Chapman. *Planning for Conjunctive Goals.* Masters's thesis MIT-AI-802, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1985.

[14] E. Charniak and D. McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley, Reading, Massachusetts, 1985.

[15] P. R. Cohen and H. J. Levesque. Persistence, intention, and commitment. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 297–340, Morgan Kaufmann, Los Altos, California, 1987.

[16] P. R. Cohen and H. J. Levesque. Speech acts and the recognition of shared plans. In *Proceedings of the Twenty Third Conference of the Association for Computational Linguistics*, pages 49–59, Stanford, California, 1985.

[17] D. D. Corkill and V. R. Lesser. The use of meta-level control for coordination in a distributed problem solving network. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 748–756, 1983.

[18] D. Davidson. *Actions and Events.* Clarendon Press, Oxford, England, 1980.

[19] L. H. Davis. *Theory of Action. Foundations of Philosophy Series*, Prentice-Hall, Engelwood Cliffs, New Jersey, 1979.

[20] P.R. Davis and R.T. Chien. Using and reusing partial plans. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, page 494, Cambridge, Massachussets, 1977.

[21] C. Dawson and L. Siklossy. The role of preprocessing in problem solving systems. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 465–471, Cambridge, Massachussets, 1977.

[22] T. Dean. *An Approach to Reasoning about Time for Planning and Problem Solving.* Technical Report 433, Computer Science Department, Yale University, New Haven, Connecticut, 1985.

[23] T. Dean. Intractability and time-dependent planning. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 245–266, Morgan Kaufmann, Los Altos, California, 1987.

[24] T. Dean. Planning and temporal reasoning under uncertainty. In *Proceedings of the IEEE Workshop of Knowledge-Based Systems*, Denver, Colorado, 1984.

[25] D. R. Dowty, R. E. Wall, and S. Peters. *Introduction to Montague Semantics. Synthese Language Library*, D. Reidel Publishing Company, Boston, Massachusetts, 1981.

[26] E. H. Durfee and V. R. Lesser. Incremental planning to control a blackboard-based problem solver. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 58–64, Philadelphia, Pennsylvania, 1986.

[27] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Increasing coherence in a distributed problem solving network. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 1025–1030, Los Angeles, California, 1985.

[28] S. E. Fahlman. A planning system for robot construction tasks. *Artificial Intelligence*, 5:1–49, 1974.

[29] R. E. Fikes. Deductive retrieval mechanisms for state description models. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 99–106, Tbilisi, Georgia, USSR, 1975.

[30] R. E. Fikes and N. J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[31] J. J. Finger. *Exploiting Constraints in Design Synthesis.* PhD thesis, Stanford University, Stanford, California, 1986.

[32] M. S. Fox. Observations on the role of constraints in problem solving. In *Proceedings of the Sixth Canadian Conference on Artificial Intelligence*, pages 172–187, Montreal, Canada, 1986.

[33] M. S. Fox. ISIS - a knowledge-based system for factory scheduling. *Expert Systems*, 1(1):24–49, 1984.

[34] M. R. Genesereth. An overview of meta-level architecture. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 119–124, 1983.

[35] M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, Los Altos, California, 1987.

[36] M. P. Georgeff. Actions, processes, and causality. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 99–122, Morgan Kaufmann, Los Altos, California, 1987.

[37] M. P. Georgeff. Communication and interaction in multiagent planning. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 125–129, Washington, D.C., 1983.

[38] M. P. Georgeff. Many agents are better than one. In *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, Morgan Kaufmann, Los Altos, California, 1987.

[39] M. P. Georgeff. A theory of action for multiagent planning. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 121–125, Austin, Texas, 1984.

[40] M. P. Georgeff and A. L. Lansky. Procedural knowledge. *Proceedings of the IEEE Special Issue on Knowledge Representation*, 74:1383–1398, 1986.

[41] M. P. Georgeff and A. L. Lansky. *A System for Reasoning in Dynamic Domains: Fault Diagnosis on the Space Shuttle.* Technical Note 375, Artificial Intelligence Center, SRI International, Menlo Park, California, 1986.

[42] M. P. Georgeff, A. L. Lansky, and M. Schoppers. *Reasoning and Planning in Dynamic Domains: An Experiment with a Mobile Robot.* Technical Note 380, Artificial Intelligence Center, SRI International, Menlo Park, California, 1987.

[43] L. Gouzenes. Strategies for solving collision-free trajectories problems for mobile and manipulator robots. *The International Journal of Robotics Research*, 3(4):51–65, 1984.

[44] C. C. Green. Application of theorem proving to problem solving. In *Proceedings of the First International Joint Conference on Artificial Intelligence*, pages 219–239, Washington, D.C., May 1969.

[45] J. Halpern. *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference.* Morgan Kaufmann Publishers, Los Altos, California, 1986.

[46] S. Hanks and D. McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 328–333, Philadelphia, Pennsylvania, 1986.

[47] D. Harel. *Dynamic Logic. Handbook of Philosophical Logic, Vol II*, D. Reidel Publishing Co., New York, New York, 1984.

[48] D. Harel. *First Order Dynamic Logic. Lecture Notes in Computer Science, 68*, Springer-Verlag, New York, New York, 1979.

[49] J. Haugeland. *Artificial Intelligence: The Very Idea.* MIT Press, Cambridge, Massachusetts, 1985.

[50] P. J. Hayes. The frame problem and related problems in artificial intelligence. In Elithorn A. and D. Jones, editors, *Artificial and Human Thinking*, pages 45–59, Jossey-Bass, San Francisco, California, 1973.

[51] P. J. Hayes. *Readings in Knowledge Representation*, chapter The Second Naive Physics Manifesto, pages 467–485. Morgan Kaufmann Publishers, Los Altos, California, 1985.

[52] B. Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321, 1985.

[53] C. A. R. Hoare. *Communicating Sequential Processes. Series in Computer Science*, Prentice Hall, Englewood Cliffs, New Jersey, 1985.

[54] R. Jeffrey. *The Logic of Decision.* University of Chicago Press, Chicago, Illinois, 1983.

[55] L. P. Kaelbling. An architecture for intelligent reactive systems. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 395–410, Morgan Kaufmann, Los Altos, California, 1987.

[56] R. Kowalski. *Logic for Problem Solving.* North Holland, New York, New York, 1979.

[57] R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Computing*, 1986.

[58] B. H. Krogh and C. E. Thorpe. Integrated path planning and dynamic steering control for autonomous vehicles. In *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, pages 1664–1669, San Francisco, California, 1986.

[59] D. T. Kuan. Terrain map knowledge representation for spatial planning. In *Proceedings of the IEEE First National Conference on Artificial Intelligence Applications*, pages 578–584, Denver, Colorado, December 1984.

[60] P. B. Ladkin. The completeness of a natural system for reasoning with time intervals. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, 1987.

[61] P. B. Ladkin and R. D. Maddux. *The Algebra of Convex Time Intervals.* Technical Report, Kestrel Institute, Palo Alto, California, 1987.

[62] A. L. Lansky. A representation of parallel activity based on events, structure, and causality. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 123–159, Morgan Kaufmann, Los Altos, California, 1987.

[63] A. L. Lansky and D. S. Fogelsong. Localized representation and planning methods for parallel domains. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-87*, Seattle, Washington, 1987.

[64] J. L. Lassez and M. Maher. The denotational semantics of horn clauses as a production system. In *Proceedings of the National Conference on Artificial Intelligence*, pages 229–231, Washington, D. C, 1983.

[65] H. J. Levesque. *Annual Review of Computer Science.* Volume 1, Annual Reviews, Inc., Palo Alto, California, 1987.

[66] V. Lifschitz. Formal theories of action. In *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, Morgan Kaufmann, Los Altos, California, 1987.

[67] V. Lifschitz. On the semantics of STRIPS. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, Morgan Kaufmann, Los Altos, California, 1987.

[68] T. Lozano-Perez. *Spatial Planning: A Configuration Space Approach.* AI Memo 605, AI Laboratory, Massachusetts Institute of Technology, Cambidge, Massachusetts, 1980.

[69] Z. Manna and R. J. Waldinger. A theory of plans. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, Morgan Kaufmann, Los Altos, California, 1987.

[70] Z. Manna and P. Wolper. *Synthesis of Communicating Processes from Temporal Logic Specifications.* Technical Report STAN-CS-81-872, Computer Science Department, Stanford University, Stanford, California, 1981.

[71] J. McCarthy. Applications of circumscription to formalizing common sense knowledge. In *Proceedings of the AAAI Non-Monotonic Reasoning Workshop*, pages 295–324, 1984.

[72] J. McCarthy. Circumscription – a form of nonmonotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[73] J. McCarthy. Programs with common sense. In M. Minsky, editor, *Semantic Information Processing*, MIT Press, Cambridge, Massachusetts, 1968.

[74] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[75] D. McDermott. Reasoning about plans. In J. R. Hobbs and R. C. Moore, editors, *Formal Theories of the Commonsense World*, pages 269–317, Ablex Publishing, Norwood, New Jersey, 1985.

[76] D. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[77] D. McDermott and E. Davis. Planning routes through uncertain territory. *Artificial Intelligence*, 22(2):107–156, 1984.

[78] R. Milner. *A Calculus of Communicating Systems. Lecture Notes in Computer Science 92*, Springer Verlag, New York, New York, 1980.

[79] R. Montague. Deterministic theories. In R. H. Thomason, editor, *Formal Philosophy: Selected Papers of Richard Montague*, Yale University Press, 1974.

[80] R. C. Moore. *Reasoning about Knowledge and Action.* Technical Note 191, Artificial Intelligence Center, SRI International, Menlo Park, California, 1980.

[81] L. Morgenstern. A first order theory of planning, knowledge, and action. In J. Halpern, editor, *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference*, pages 99–114, Morgan Kaufmann Publishers, Los Altos, California, 1986.

[82] J. K. Myers. Multiarm collision avoidance using the potential-field approach. *Society of Photo-Optical Instrumentation Engineers*, 580:78–87, 1985.

[83] A. Newell and H. A. Simon. GPS, a program that simulates human thought. In E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–293, McGraw-Hill, New York, 1963.

[84] N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Company, Palo Alto, California, 1980.

[85] N. J. Nilsson. *Shakey the Robot.* Technical Note 323, Artificial Intelligence Center, SRI International, Menlo Park, California, 1984.

[86] N. J. Nilsson. *Triangle Tables: A Proposal for a Robot Programming Language.* Technical Note 347, Artificial Intelligence Center, SRI International, Menlo Park, California, 1985.

[87] H. Nishimura. Descriptitively complete process logic. *Acta Informatica*, 14:359–369, 1980.

[88] E. P. D. Pednault. Solving multiagent dynamic world problems in the classical planning framework. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 47–82, Morgan Kaufmann, Los Altos, California, 1987.

[89] E. P. D. Pednault. *Toward a Mathematical Theory of Plan Synthesis.* PhD thesis, Department of Electrical Engineering, Stanford University, Stanford, California, 1986.

[90] R. Pelavin and J. F. Allen. A formal logic of plans in a temporally rich domain. *Proceedings of the IEEE, Special Issue on Knowledge Representation*, 74:1364–1382, 1986.

[91] M. E. Pollack. *Inferring Domain Plans in Question Answering.* PhD thesis, Computer Science Department, University of Pennsylvania, Pittsburgh, Pennsylvania, 1986.

[92] M. E. Pollack. A model of plan inference that distinguishes between the beliefs of actors and observers. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 279–295, Morgan Kaufmann, Los Altos, California, 1987.

[93] D. L. Poole, R. G. Goebel, and R. Aleliunas. *Theorist: a logical reasoning system for defaults and diagnosis.* Springer-Verlag, New York, New York, 1986.

[94] A. N. Prior. *Past, Present and Future.* Clarendon Press, Oxford, England, 1967.

[95] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[96] J. S. Rosenschein. *Rational Interaction: Cooperation among Intelligent Agents.* PhD thesis, Computer Science Department, Stanford University, Stanford, California, 1986.

[97] J. S. Rosenschein. Synchronization of multi-agent plans. In *Proceedings of the National Conference on Artificial Intelligence*, pages 115–119, Stanford, California, 1982.

[98] J. S. Rosenschein and M. R. Genesereth. *Communication and Cooperation.* Technical Report 84-5, Heuristic Programming Project, Computer Science Department, Stanford University, Stanford, California, 1984.

[99] S. J. Rosenschein. Plan synthesis: a logical perspective. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 331–337, Vancouver, British Columbia, 1981.

[100] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. In *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pages 412–430, Stanford, California, 1973.

[101] E. D. Sacerdoti. *A Structure for Plans and Behaviour.* Elsevier, North Holland, New York, 1977.

[102] F. Sadri. Representing and reasoning about time and events: three recent approaches. Department of Computing, Imperial College, London, 1986.

[103] Y. Shoham. Chronological ignorance: time, nonmonotonicity, necessity and causal theories. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 389–393, Philadelphia, Pennsylvania, 1986.

[104] Y. Shoham. What is the frame problem. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 83–98, Morgan Kaufmann, Los Altos, California, 1987.

[105] M. Stefik. Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16(2):111–140, 1981.

[106] M. Stefik. Planning with constraints (MOLGEN: Part 2). *Artificial Intelligence*, 16(2):141–170, 1981.

[107] H. R. Strong. Translating recursive equations into flowcharts. *Journal of Computer and System Sciences*, 5(3):254–285, 1971.

[108] C. J. Stuart. *Synchronization of Multiagent Plans Using A Temporal Logic Theorem Prover.* Technical Note 350, Artificial Intelligence Center, SRI International, Menlo Park, California, 1985.

[109] G. J. Sussman. *A Computational Model of Skill Aquisition.* Technical Report AI TR-297, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1973.

[110] A. Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 888–893, Cambridge, Massachusetts, 1977.

[111] A. Tate. Goalstructure — capturing the intent of plans. In *Proceedings of the Sixth European Conference on Artificial Intelligence*, pages 273–276, Pisa, Italy, 1984.

[112] A. Tate. Planning in expert systems. In *Alvey IKBS Expert Systems Theme—First Workshop*, Cosener's House, Abingdon, Okford, England, March 1984. Also D.A.I. Research Paper 221, University of Edinburgh.

[113] A. Tate. INTERPLAN: *A Plan Generation System which can Deal with Interactions Between Goals.* Memorandum MIP-R-109, Machine Intelligence Research Unit, University of Edinburgh, Edinburgh, Scotland, 1974.

[114] R. H. Thomason. The context-sensitivity of belief and desire. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 341–360, Morgan Kaufmann, Los Altos, California, 1987.

[115] J. van Bentham. Tense logic and time. *Notre Dame Journal of Formal Logic*, 25(1):1–16, 1984.

[116] S. Vere. Planning in time: windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(3):246–267, 1983.

[117] R. Waldinger. Achieving several goals simultaneously. *Machine Intelligence*, 8:94–136, 1977.

[118] D. H. D. Warren. WARPLAN: *A system for generating plans.* Technical Report, University of Edinburgh, Edinburgh, Scotland, 1974.

[119] L. P. Wesley, J. D. Lowrance, and T. D. Garvey. *Reasoning about Control: An Evidential Approach.* Technical Note 324, Artificial Intelligence Center, SRI International, Menlo Park, California, 1979.

[120] R. Wilensky. Meta-planning: representing and using knowledge about planning in problem solving and natural language understanding. *Cognitive Science*, 5:197–233, 1981.

[121] D. E. Wilkins. Domain independent planning: representation and plan generation. *Artificial Intelligence*, 22:269–301, 1984.

[122] D. E. Wilkins. *Hierarchical Planning: Definition and Implementation.* Technical Note 370, Artificial Intelligence Center, SRI International, Menlo Park, California, 1985.

[123] D. E. Wilkins. Recovering from execution errors in SIPE. *Computational Intelligence*, 1:33–45, 1985.

[124] D. E. Wilkins. *Using Causal Rules in Planning.* Technical Note 410, Artificial Intelligence Center, SRI International, Menlo Park, California, 1987.